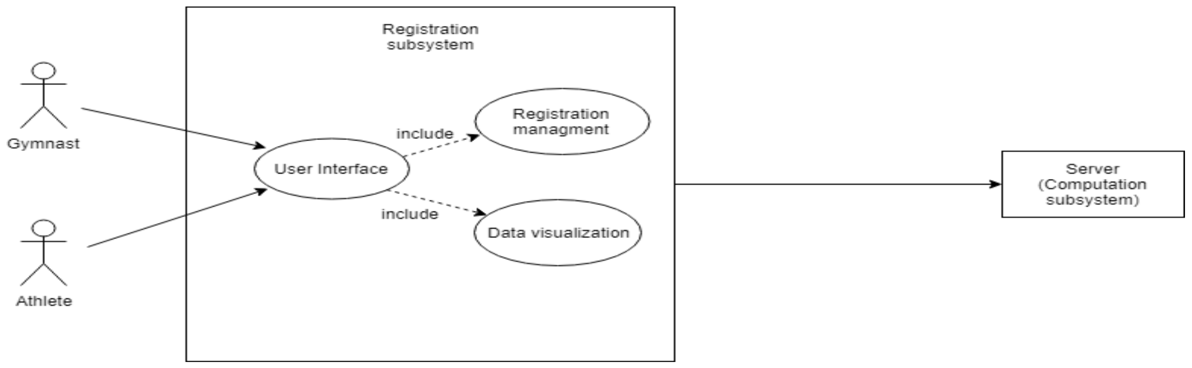
Waeel Alhrazy(wa22bq)
Alaa Alwan(aa223fw)

Architecture Decisions for Gymnast League System

Task 1: Stakeholder: Gymnastics League Board of Directors

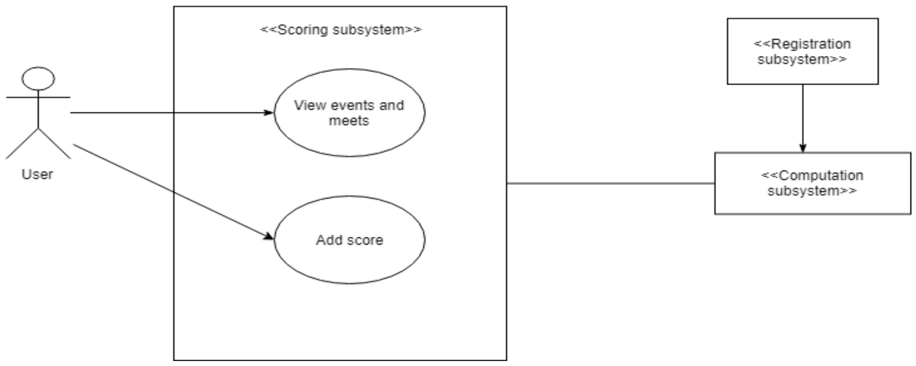
Based on the preceding two preliminary assignments that were generated prior to this, it is important that we isolate the necessary factors that would be highlighted in this report. Our system is divided into subsystems and to explain this complexity of the system structure we should point out sour major component (Client and Server) used to model our system each component has its components. We will use UML to model our cases.

1- Registration subsystem: The system support data input of all information about teams, gymnasts and judges.



It also manages the gymnast and meet. Information and rules will be processed for further components, where it manages the rules of teams, meets, judges, and scores. This system is responsible for handling the personal data of gymnasts, judges, athletes and setting the rules to start the season league. Registration is a major component for the system as it stores the inputs and transfer it to Computation subsystem that compute the meets and the whole computation according to the inputs it receives. This dependency between Registration and Computation is major where users would register and view their account and results through UI and this decision was made according to the requirements.

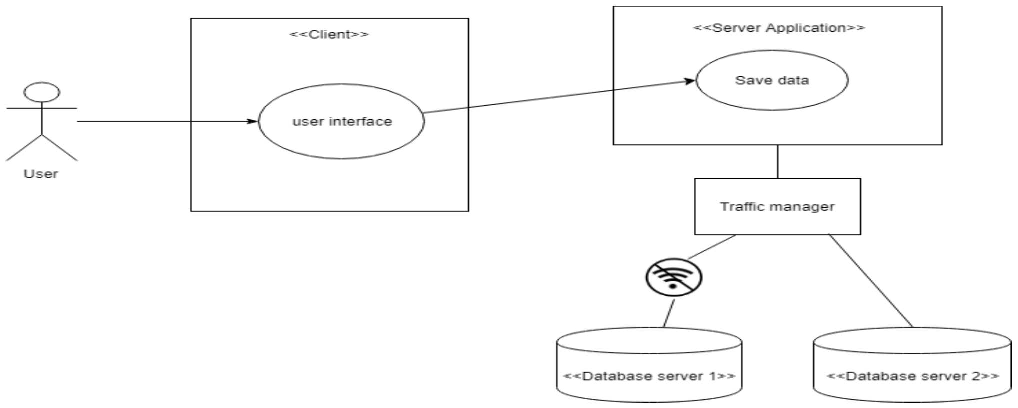
2- Scoring subsystem: responsible for keeping and counting the score of each competitor, team, competition and meet all type of scores. The data will be transferred to the server.



The judges will provide the needed input and then will send it back to the computation subsystem to save it. Now the computation component shall be able to calculate the meet scores for each team for the events. This is the most common process of calculating scores in gymnastics systems so we planning to build or software according to it as its efficient. The score subsystem dependent on the Computation subsystem in server application side to be fed by the teams' information and events then record the scores inputted by the judge and send it back to Computation subsystem again to save it in databases. The teams are also able to go and view their meets, events and score list through score management subsystem.

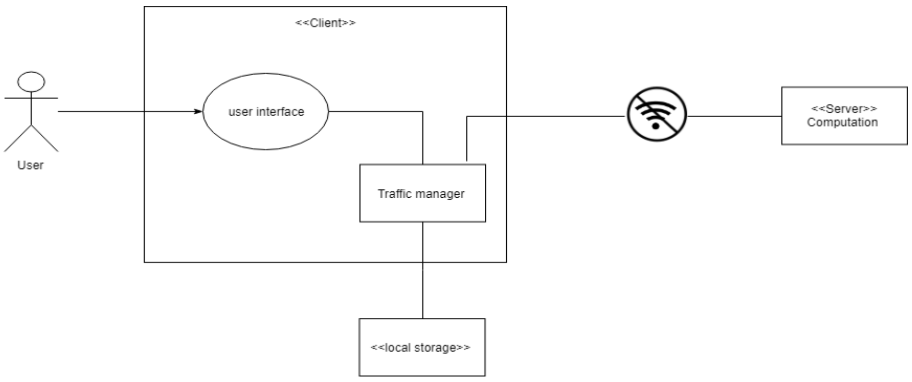
3- Computing subsystem: this subsystem is responsible of computing assigning events and meets for teams to compete according to information and rules it receives from registration subsystem, then it share those information with score subsystem so the judges be able to input a score to the meets then send updated data to computation subsystem to save it in the database. We decided to let the computation subsystem to play the proxy between the client and database to control the data given from the client side. Keep record of the seasonal standings is saved in the database and computing subsystem is responsible to call it whenever the user requires it.

4- The system supports multiple database servers to avoid single point of failure in the application server.



If one server encounters a critical error, service automatically switches to another server, and end users are not prevented from using the application. Single point of failure is undesirable to our system where data availability is priority so upon server failure there is another server to take over and save the data from getting lost this is one service beside the other functionalities. Such redundancy will keep the system operation continuously.

5- The system saves all data in a local storage whenever the connection client and server are down until the connection is back again.

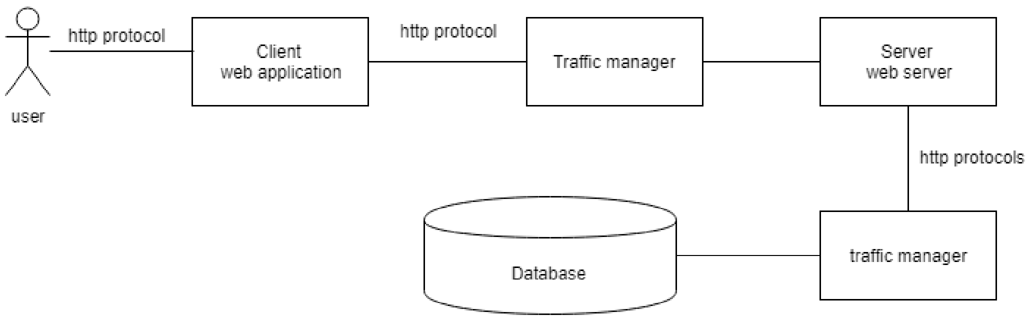


When the data is processed to application server and suddenly a connection failure occurs the data will be redirected to the local storage which keep the data till the connection is back. The traffic manager will take responsibility of operating this task while the system is running and make sure the data goes handled between local storage to server or from client to server depending on the connection status. This will protect the data from being lost if any connection errors happen. This will increase the system availability and provide automatic failover when an endpoint goes down.

Structure, software and hardware

The system is designed to run as a web application where it will be based in a server. All the computing and calculation will be done on the server-side, this way we guarantee the users can access to the application and interact on any gadget from any place though a web browser instead of downloading an app to every device to use the application (the interaction among users are different according to their role). Our system will have two external servers which operates as databases for storing the data, we decided to have two server to avoid single point of failure. In the client-side we would also save data for persistency and call local storage where data will be stored as JSON file in case the server is down, we choose JSON technology because the data file formats are text-based and primarily used for transmitting data. The communication between client and server based on http protocols. The connection is made through the internet via https requests.

The figure below show the visualization of the structure of system software and hardware



Discussion task 1

The Board of Directors are mainly concern about the interface of the system and the functionality views that would impact on their business decisions, so we have discussed our system functionality focusing on the concerns of the board as their knowledge of the software development background is simple. Therefore, we used the UML diagrams to view the functionality of the system and address the requirements. We also elaborate how we divided our system and what are the main subsystem and their functionality. Finally, we also explained the software and hardware that we are going to use to build the system using common reasoning to describe the complexity of the technologies and the system.

Task 2: Solution Architects/Designers

The system we purposed is a web-application which is accessible through any device and flexible. The server approach increases the scalability of the system as the system could be updated and modified to convoy state-of-the-art technology. The web-application will have also an authentication system to give users functionality according to their role such as judges can view meets and assign scores, teams view their meets schedule and results, gymnasts input information, rule and type of meets. The system is divided into Server-Client and each side has its components that represents the subsystem of the system. Our client has the user interface or web interface that view the registration and schedule of the meets, then we have three servers application server, and two databases servers, as well we have two traffic manager components to handle the data redirecting between client and server also application server and database.

Client:

This or web-application which run on a web-server, the user interface is user-friendly so we will use javascript as a programming language and React framework, for designing the interface HTML5 and CSS3 are the best technologies that are used most widely and compatible with JS to develop the best interface and variety of design options it provides. Bootstrap4 is also a good framework to develop our css in designing the interface to ease and view our design options before deploying, there are similar alternative frameworks but Bootstrap is widely used.

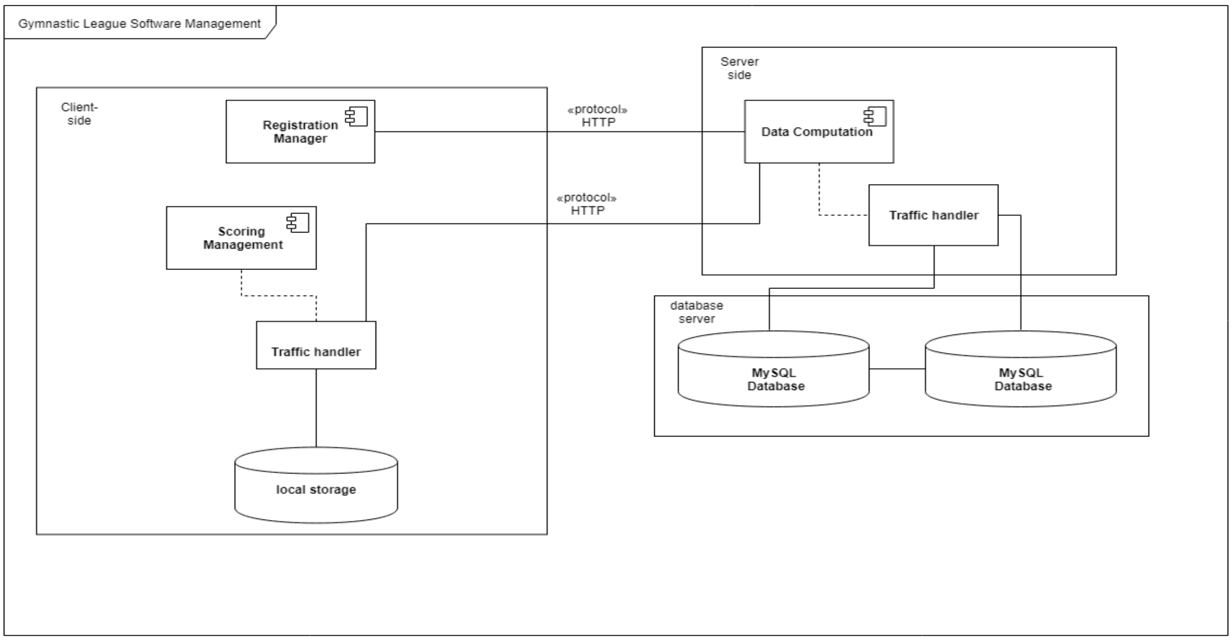
Application server:

Web-server where the application can run and compute the data it receives from the users inputs and responds to it, it also store the data into the database and retrieve data when it needed. NodeJS platform with JavaScript programming language we choose node.js because its an open source framework that will help developers to apply changes in flexible way and plus it's a runtime built on google chrome engine which is widely used and easy to develop. Alternative could be GOLang which is a new google language developed primarily for web frameworks, however Node.js is reliable and more developers can develop using it since it has been used for years. We would use the Express framework for security and for easier implementation and deployment because we are using NodeJS. The application server will be responsible for communicate with Registration subsystem from client side to get data and then after it compute it will be sent again to client side but this time to the scoring management to share the event and meets and get scores from client side to the database. All the communications are through HTTP requests. Websockets would be used to communicate with the client, for updating instantly the information in the client side. The data would be transferred in JSON form. The application server will communicate with the database through traffic manager which is a class in JS that is responsible of the data traffic during offline and online to save data from getting lost.

Database server

These servers are responsible to receive data and store it, those are remote database which communicate with the application server through webhooks, and exchange MySQL queries with it. The idea of having two database it to solve the single point of failure and rescue data from being lost due to any connection failure during the transfer of data. Both database servers run on Linux with Oracle software to gain better persistency and stability. The database server has database component that stores all the data, and keep records of the seasonal league, we decide to use MySQL which is suitable to deal with big analytical data, optimization and data modelling.

The figure below view the system to developers using component diagram:



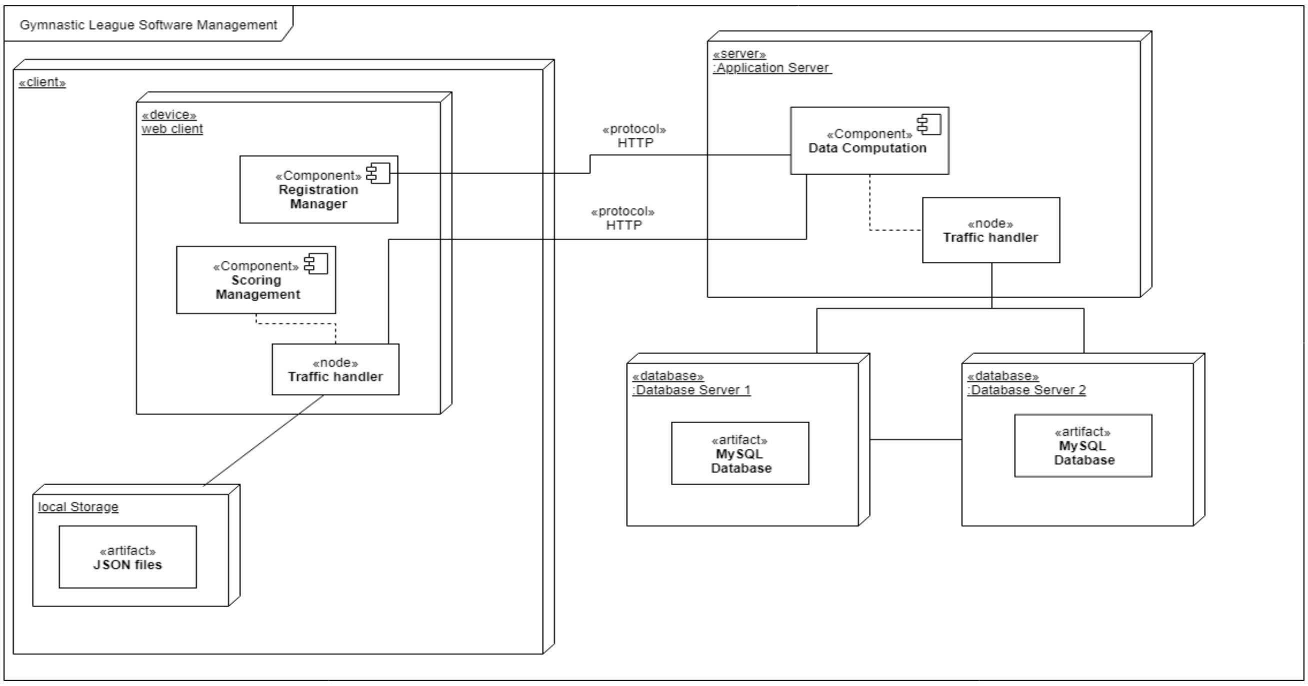
Discussion task 2

In this task we explained the software that is to be used and implementation of the system to one important stakeholder the developers and designers. We choose the software according to our resources and researches that will be suitable to build this system taking in consideration valid quality attribute such as scalability, availability and maintainability which is mainly what the developers are concern about, we also addressed the subsystems of the system and what tools are proper to use with providing some alternatives. We suggested a real time web application which is suitable for multiple user in our case (Judge, gymnast, team) instead of an installed application that we cost time and effort to have it on many gadgets at the same time. We also addressed how the components will be interact with each other and in what capacity, this will give the developers a better understanding of the system for future development in requirements.

<u>Task 3: Operations/Testers</u>

Testers are interested in the system components integration, also if the designed fulfils the format requirements and the architecture. System deployment diagram will give a proper view to address the concerns of the stakeholders in proper manner. In this section, we will discuss the deployment view and the physical arrangement of the product software components. As we mentioned in our design that the system is Client-Server approach and to illustrate the how it would be deployed, we use UML deployment diagram to represent how the components are connecting and communicating with each other.

Software deployment diagram


Gymnastic League Software Management
«client»
«device» web client
«Component» Registration Manager
«Component» Scoring Management
«node» Traffic handler
local Storage
«artifact» JSON files
«protocol» HTTP
«protocol» HTTP
«server» :Application Server
«Component» Data Computation
«node» Traffic handler
«database» :Database Server 1
«artifact» MySQL Database
«database» :Database Server 2
«artifact» MySQL Database

The above diagram includes both client-side and server-side components to show the whole system and components connect with each other. The diagram also shows the overall deployment of the system.

Client-side

Our system operates on PC or Mobile device that has access to the web interface. The system accessed through a web browser that communicates to our application server. The Web interface is a web application implemented using Node.js platform. Web client has three components:

- Registration manager that is responsible for data inputs.
- Score management responsible for data input and view.
- Traffic manager node for redirecting data.

Local storage a file created in the client hard drive to save data files received by traffic handler in case of the connection failed between client and server. The data file formats are JSON files which are text-based and primarily used for transmitting data. The communication between client and server based on http protocols.

Server-side

Application server is web-server that operates on operating system Linux and Apache due to their considerable flexibility, stability and security. The server components will be implemented using JavaScript, since it’s object-oriented language and Javascript has excellent open source-libraries. Also, it works on Node.js . Data computation component will act as a module to communicate between the client and database. Data computation component also responsible for data calculation; data inputs received from client side and handled to give outputs after parsing and manipulate the input data.

Traffic handler is implemented in server side as well to handle the data traffic between Application server and database server, it's basically responsible for redirecting data if the connection failed between either of the database servers in Data warehouse.

Data warehouse consist of Database server 1 which is our primary database and Database server 2 is our secondary database. Both database servers run on Linux with Oracle software to gain better persistency and stability. The database server has database component that stores all the data, we decide to use MySQL which is suitable to deal with big analytical data, optimization and data modelling. If the traffic handler could not connect to the first database due to connection failure, it will redirect the data to database server 2, this decision was made to avoid one single point failure in the server side.

Discussion task 3

In this task, we explain the details of the product to show the physical aspects of the deployment level. Testers are important stakeholders that should understand the components of the system and apply practical test plans to verify a successful deployment, therefore they are interested in the physical components and arrangement of the system. Operation stuff has the same concerns as testers, which is the system deployment. UML deployment diagram illustrates the physical layer of the architecture decisions used to achieve the aspect of the system. Basically, this diagram helps testers to deploy the system understanding how it supposes to function, and it sorts as a communication method between operation stuff and testers to deploy the system same as it was designed in earlier stages to fulfil the requirement.

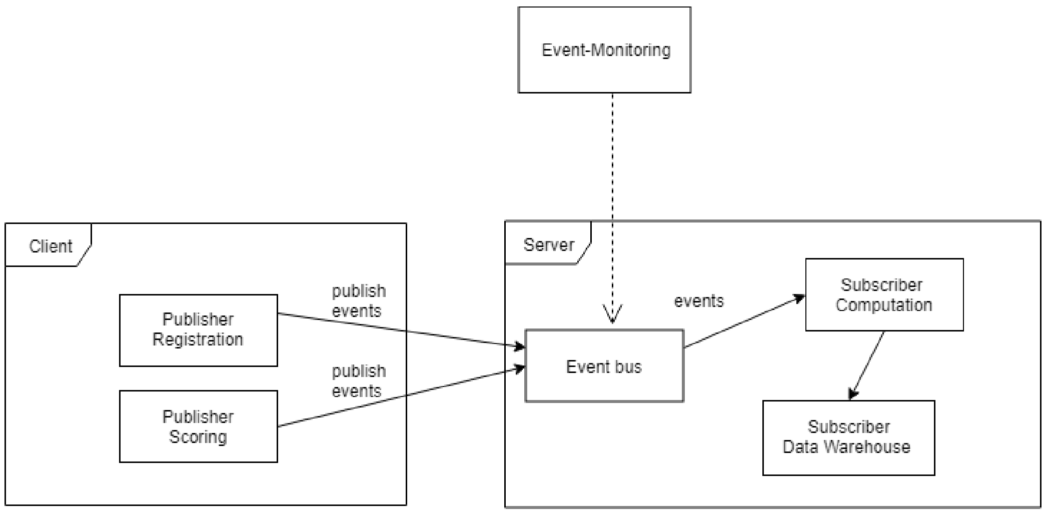
Task 4: Event Monitoring:

Event-logging

According to our decision, software architecture is Client-Server pattern. As a result, to achieve the event-logging concern, the application should include a broker called bus-event that will be responsible to carry the events data from one component to another. Bus-event will communicate components that publish events to components that subscribed to receive published events; in our system, it will handle the communication between the client and server to monitor the events. In stakeholder requirement, the system could be accessed by multiple clients and that would cause a problem to our server to handle all these event logging from multiple clients. Therefore, with Event-bus subsystem we can add multiple clients at the same time to be hosted on the same server and will take care of handling all clients that want to communicate to the server database. The components will interact without being aware of each other, and without depending on each other and that to maintain loose coupling. Moreover, we can monitor the events through the bus-even subsystem and handle any connection faults. We can check the logging and detect fail logs and fix it and this achieves availability as well.

Event-bus will achieve synchronous communication between local storage in client-side and data warehouse in server-side. Subscribers in the server-side register for specific event and it is triggered when that event is published from the Publisher, we implement a listener method that runs synchronously to handle the communication. Event-logging will achieve high decoupling as well. First, on the client-side the components are registered as sources so they could publish events to the server side and wait for responses. The data will be sent as messages to the bus-event which is a broker between the client and server, bus-event is a channel that will receive the messages published and then transfer them to the registered components from server-side which are called subscribers. Those subscribers request specific events that they are interested in to receive from client-side sources, so the bus-event duty here to distribute every published event to the subscribers that requested the event and then take the response back to publishers in client-side. In the subscribers, listeners will be implemented to listen to the published events occurs in the bus-event.

In the figure below, we show the mechanism of event-logging between client and server through our broker Event-bus.



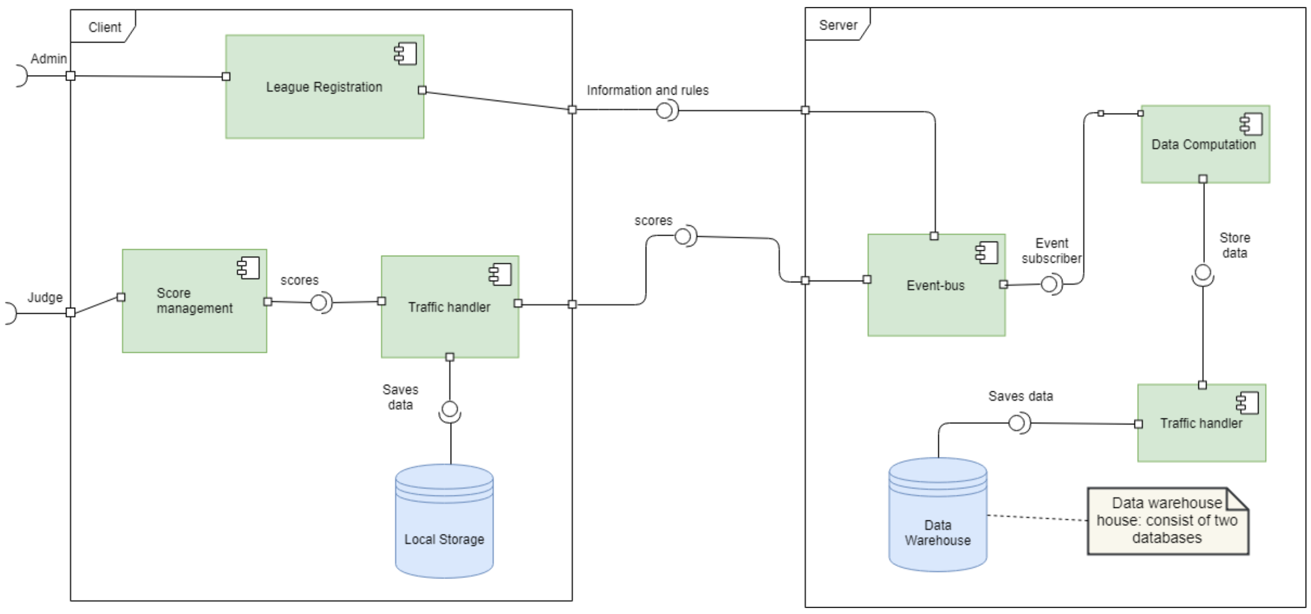
As we explained previously on the process of how we achieve event-logging by adding event bus component as a broker between the publishers and subscribers as shown in the figure above. Event-bus now can monitor all the events occurs, as we can decide what type of data should be entered and discard the unwanted once.

Waeel Alhrazy(wa22bq)
Alaa Alwan(aa223fw)

Architecture design

In our system, client-side we have two subsystems and that are the sources that publishes events as messages to the bus-even component. One subsystem in server-side, that will be the subscriber, those components registered in the broker to receive the published events. All these subsystems will have listeners or observers to listen to the events they have registered for.

The figure below views our software architecture after we add the event-bus subsystem to achieve event-logging concern.



Client-side:

- Registration subsystem (Publisher): responsible to send information and rules through its interface to server-side, Registration publishes event to the server-side for computation, the event transfer through the middleware channel, this channel will transmit the event around the components until the Computation subsystem listener notice it and request it from the middleware which will deliver it to the destination.
- Scoring subsystem (Publisher): responsible for managing the scores inputs. The score as well will send data to middleware, but not directly, as it has to be through traffic manager that transfer the event to the middleware. Then the same process done to reach Computation subsystem in server-side.
- Traffic handler: responsible of redirecting data between Local storage and *Data warehouse* in server side. It detects lost connection and switching between the server and the local storage. The events that are published from scores will be taken to the broker.
- Local storage: responsible of storing data received from traffic manager to save it from getting lost. In case of any, malfunction in the server-side Local storage stores the data till server is fixed.

Server-side:

- Data computation subsystem (Subscriber): responsible for the computation (setting up) of the meet league; computes the teams, rules, games, scores etc. also, it's responsible to transfer/retrieve data to the Data warehouse. When the published events channelled through the event-bus component the listeners in data computation subsystem will notice there is an event they are interested in and call it. This subsystem has two responsibilities after listening to an event (receiving data):
  - To send a response back to Registration subsystem in client-side; the
  - To send a response back to Score subsystem in client-side.

  Responses could be a confirmation message, computation result such as results/ information or records of teams/schedules/meets etc.

- Event-bus: responsible for handling events when triggered and distribute them between components in client and server sides.
- Traffic handler: responsible for transfer data between Data computation and *Data warehouse* in server-side. It redirects data between the two databases in the data warehouse. It detects lost connection and switching between the database 1 and database 2 in data warehouse.
- Data warehouse: responsible for saving all types of data from client-side and Data Computation subsystem as well. It also responsible for record keeping for the schedule of meets, assigned judges to the events and seasonal standings. Besides, it saves the score for every competitor.

Motivate Decision:

We have decided to reuse the RabbitMQ software architecture which is Event-based architecture because it provides us with a simple solution in parsing the events and manipulate them according to our need. The event-bus pattern is suitable to our system for high decoupling, loose coupling and easy to implement and update according to our need. Traffic handler is used to manage the problem of disconnection between client and server and maintain data persistence. Data warehouse has two databases to avoid one single point failure in the server side.

Discussion task 4

Our system was designed in an earlier stage without considering event handling or logging, in general, we dependent on traffic manager to deal with the load of the data and manage to redirect the data incase client lost any communication with the server for any reason. Therefore, in this task we added an additional subsystem as a broker to deal with the data before it reaches its destination and monitor it in case, we have any faults that would cause failures to the subsystems in server-side. Event bus used in bigger systems where it could be a second server(middleware) to monitor the data before it reaches to the server-side. However, in our case it's sufficient to have it as a subsystem in our server-side to not add more complexity to the system.